Eliminating code coverage differences from large-scale programs

Ferenc Horváth

During software development special activities are done to keep the quality of the software while the requirements and the code are constantly changing. This includes, white-box test design, massive regression testing, selective retesting, efficient fault detection and localization, as well as maintaining the efficiency and effectiveness of the test assets on a long term [1]. These activities are usually based on *code coverage*, a test completeness measure, therefore, code coverage measurement is an important element both in industrial practice and academic research. Obviously, inaccuracies of a code coverage tool sometimes do not matter that much but in certain situations they can lead to serious confusion. For Java, the prevalent approach to code coverage measurement is to use bytecode instrumentation due to its various benefits over source code instrumentation. However, there can be differences in the list of covered items reported by the two approaches, and these differences have influence on the information derived from them. Tengeri *et al.* [2] investigated this two types of code coverage measurement on Java systems: they analyzed the results of method level coverage measurements on Java programs and found that there are many deviations in the raw coverage results due to various technical and conceptual differences of the instrumentation methods. Similar studies exist in relation to branches and statements [3], where the authors investigated how the differences can impact further activities.

In this paper, I extend the work of Tengeri *et al.* [2] and present the results of an empirical study conducted on eight large-scale programs, concentrating on how the tools can be configured and the results be filtered so that the causes are eliminated and the differences in the coverage results are alleviated as much as possible. In addition, I present my experiences on how big difference remains after eliminating tool-specific differences, which can be possibly attributed to the differences in the fundamental approach, that is, bytecode vs. source code instrumentation. The results indicate that in the programs with the biggest overall coverage difference, about 10% of the methods are falsely reported as not covered, and that when all factors including not recognized and fewer times covered cases are taken into account, the difference is much more emphasized. On average, 20% of all methods of the eight programs have false coverage data, when investigated more closely.

References

[1] L. S. Pinto, S. Sinha, and A. Orso, "Understanding myths and realities of test-suite evolution", in *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*. ACM, 2012, pp. 33:1–33:11.

[2] D. Tengeri, F. Horváth, Á. Beszédes, T. Gergely, and T. Gyimóthy, "Negative effects of bytecode instrumentation on java source code coverage", in *Proceedings of the 23rd IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER 2016)*, Mar. 2016, pp. 225–235.

[3] N. Li, X. Meng, J. Offutt, and L. Deng, "Is bytecode instrumentation as good as source code instrumentation: An empirical study with industrial tools (experience report)," in *Software Reliability Engineering (ISSRE), 2013 IEEE 24th International Symposium on*, Nov 2013, pp. 380–389.